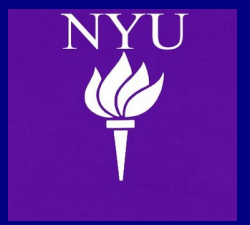
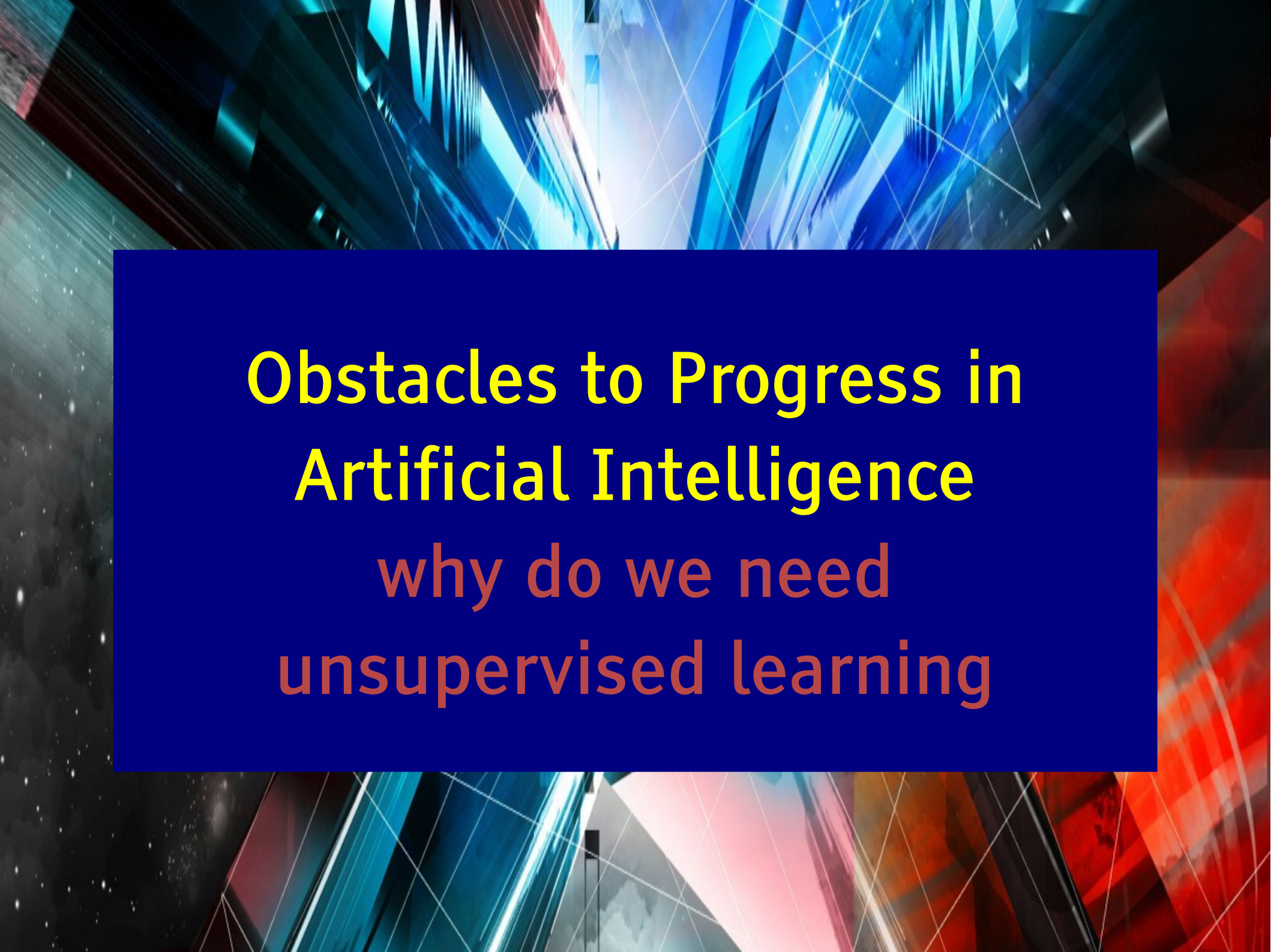
Unsupervised Learning part 1

Yann Le Cun
Facebook AI Research,
Center for Data Science, NYU
Courant Institute of Mathematical Sciences, NYU
http://yann.lecun.com




Obstacles to Progress in Artificial Intelligence
why do we need unsupervised learning

Three missing pieces for AI (besides computation)

Integrating Representation/Deep Learning with Reasoning, Attention, Planning and Memory

- A lot of recent work on reasoning/planning, attention, memory, learning "algorithms".
- Memory-augmented neural nets
- "Differentiable" algorithms

Integrating supervised, unsupervised and reinforcement learning into a single "algorithm".

- Boltzmann Machines would be nice if they worked.
- Stacked What-Where Auto-Encoders, Ladder Networks….

Effective ways to do unsupervised Learning

- Discovering the structure and regularities of the world by observing it and living in it like animals and humans do.

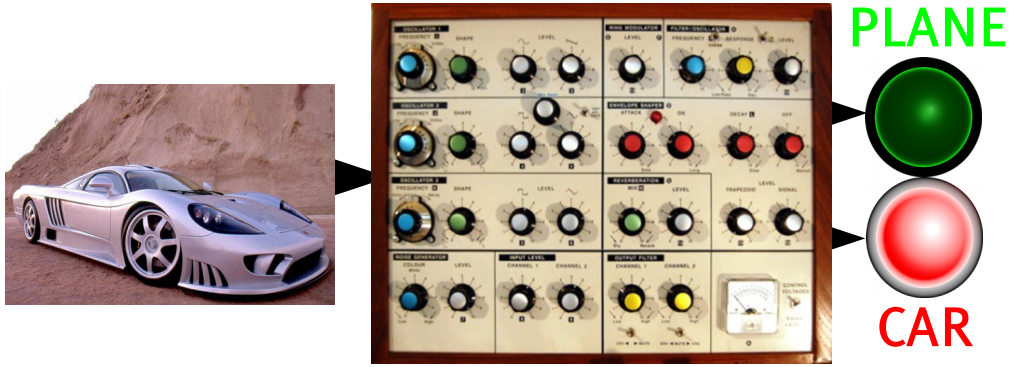
Three Types of Learning

Reinforcement Learning

- The machine predicts a scalar reward given once in a while.
- A few bits for some samples

Supervised Learning

- The machine predicts a category or a few numbers for each input
- 10→10,000 bits per sample



Unsupervised Learning

- The machine predicts any part of its input for any observed part.
- Predicts future frames in videos
- Millions of bits per sample

How Much Information Does the Machine Need to Predict?

Y LeCun

Reinforcement Learning (cherry)

- The machine predicts a scalar reward given once in a while.
- **A few bits for some samples**

Supervised Learning (icing)

- The machine predicts a category or a few numbers for each input
- **10→10,000 bits per sample**

Unsupervised Learning (cake)

- The machine predicts any part of its input for any observed part.
- Predicts future frames in videos
- **Millions of bits per sample**

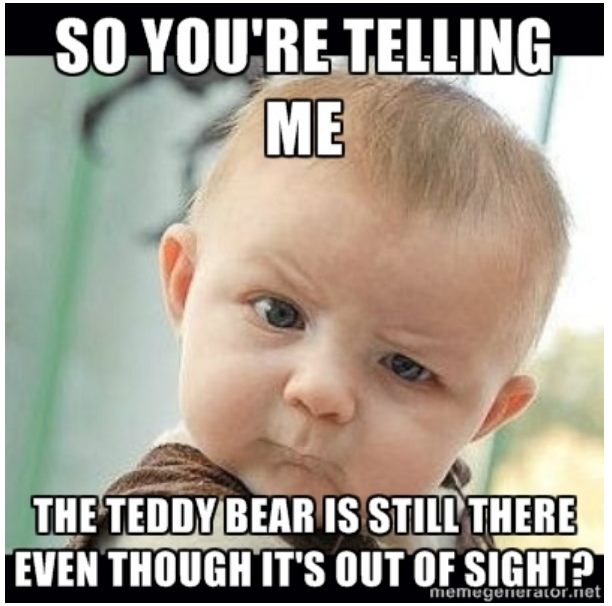
Unsupervised Learning is the "Dark Matter" of AI

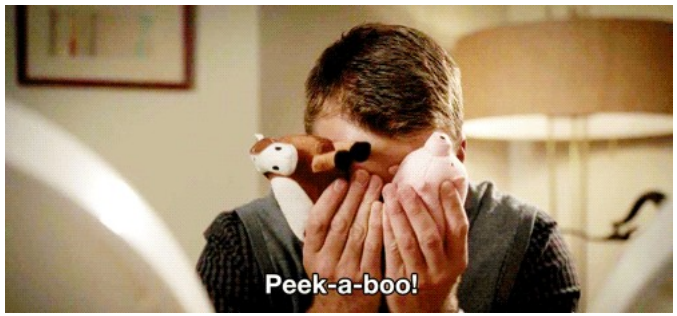
Y LeCun

Most of the learning performed by animals and humans is unsupervised

We learn how the world works by observing it

- We learn that the world is 3-dimensional
- We learn that objects can move independently of each other
- We learn that object permanence
- We learn to predict what the world will look like one second or one hour from now.





We build a model of the world through predictive unsupervised learning

This predictive model gives us "common sense"

Unsupervised learning discovers regularities in the world.

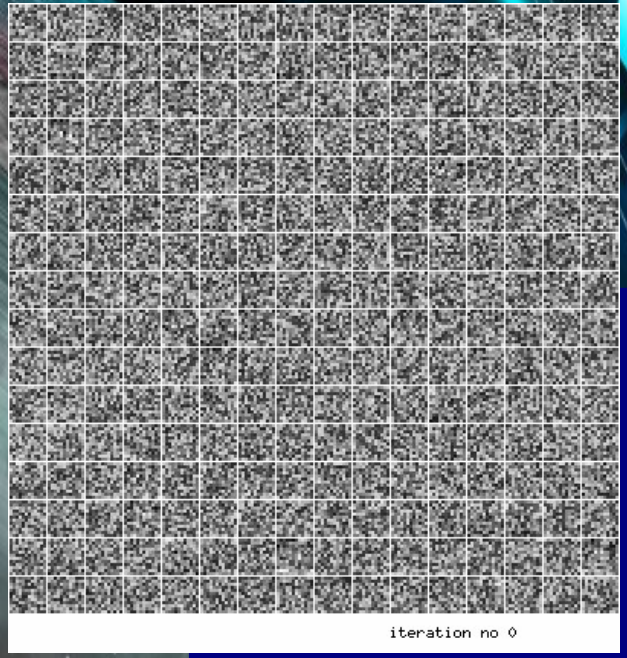
Common Sense through Unsupervised Learning

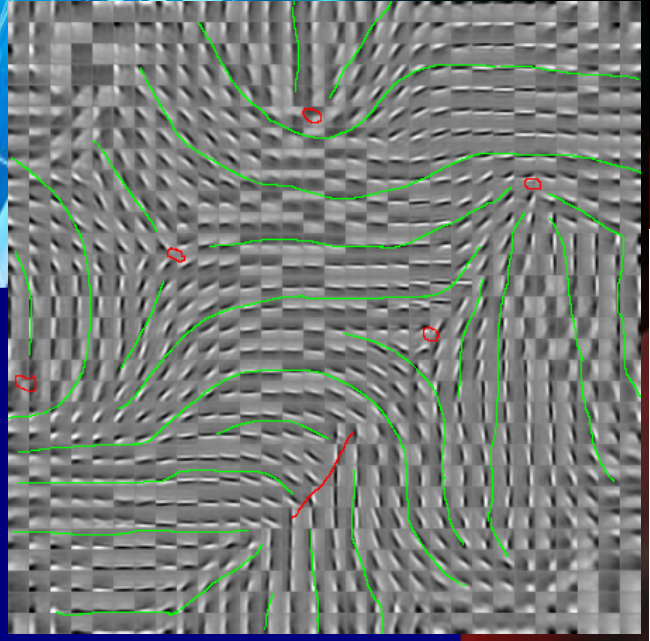
Learning a predictive model of the world gives us common sense.

If I say: "Bernhard picks up his bag and leaves the room"

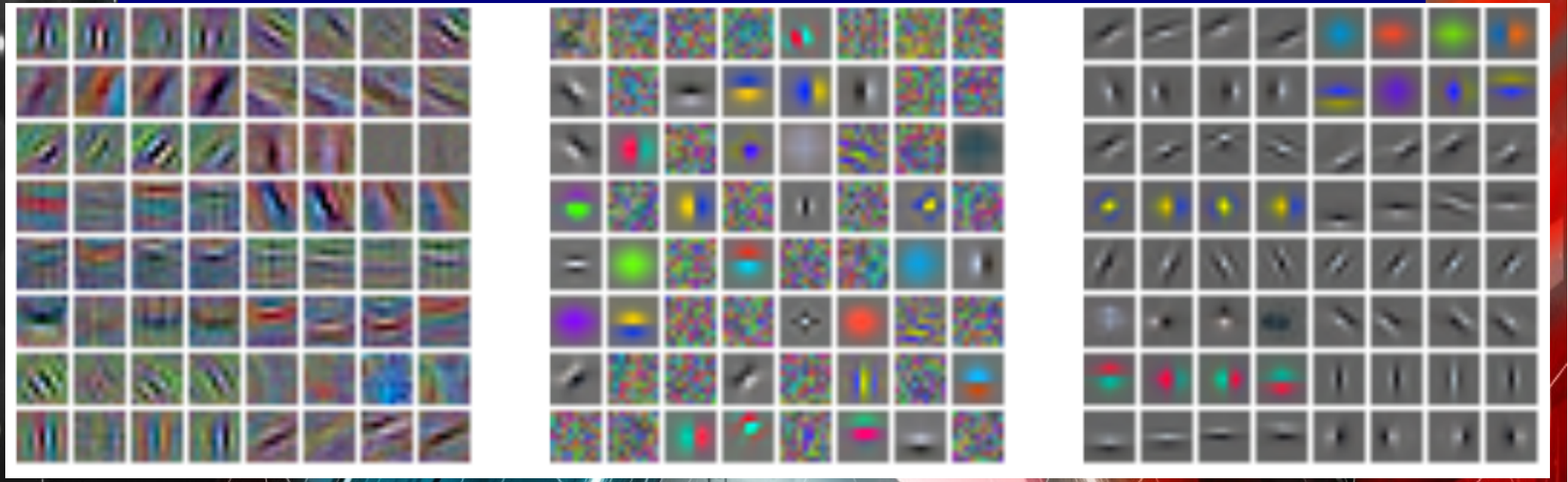
You can infer:

- Bernhard stood up, extended his arm, walked towards the door, opened the door, walked out.
- He and his bag are not in the room anymore.
- He probably didn't dematerialize of fly out.





Unsupervised Learning



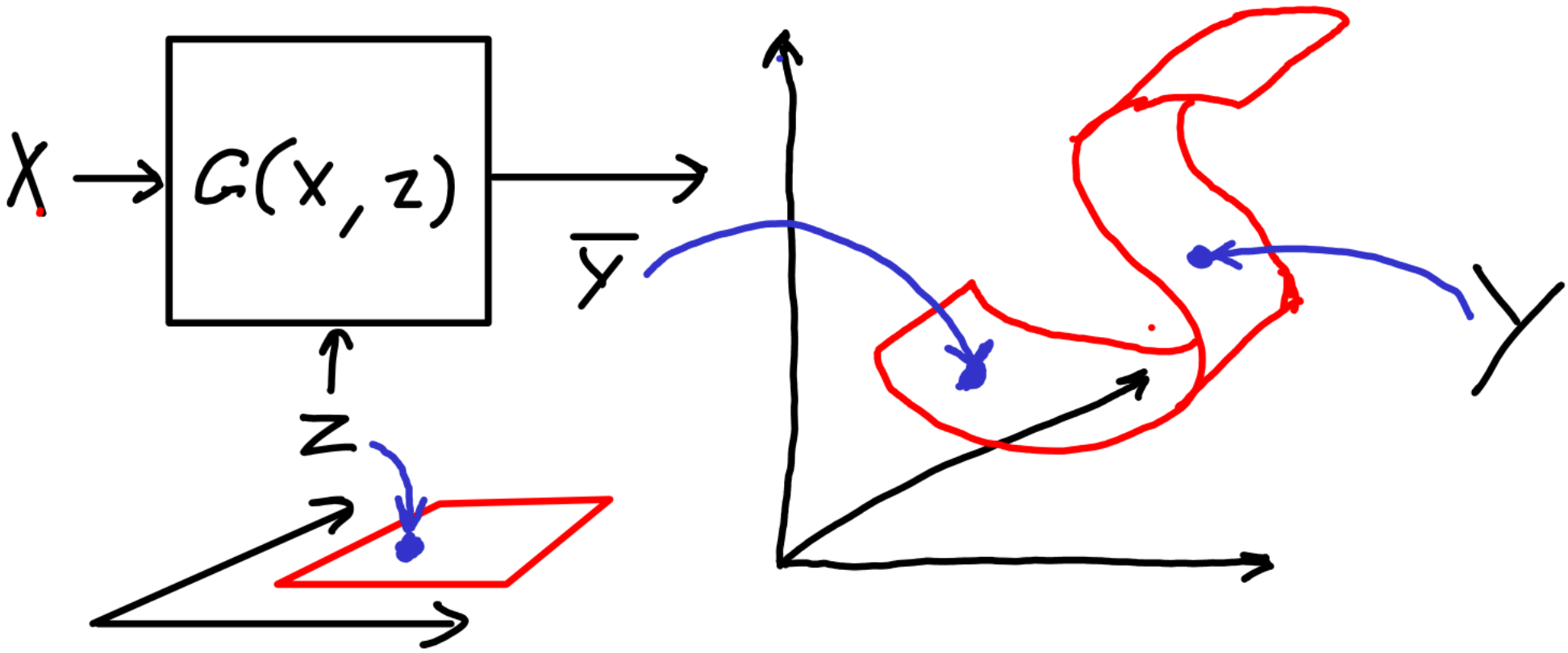
How do we do unspervised learning?

- **Unsupervised learning can be based on reconstruction or prediction**
  - Reconstruction is just prediction with delay 0
  - It seems qualitatively different from prediction, but it's not
  - Because a good way to predict the future is to copy the present
- **How do we deal with the fact that the world is only partially predictable?**
  - Model the future as a **distribution**: **hahaha, good luck with that!**
  - Modeling a distribution works when the world is discrete (e.g. words)
  - It doesn't work when the world is continuous or compositional.
- **Even if the world were noiseless and quasi deterministic, our current methods would fail**
  - Imagine the world follows a trajectory on a low-dimensional manifold with a few discrete choices at certain places.
  - Our current methods would still fail.
  - Our failure to do unsupervised learning has nothing to do with our inability to model **probability distributions** in high-dim spaces.
  - **It has to do with our inability to capture complex regularities.**
  - **I'm allergic to sampling anyway.**

Invariant Prediction: the output is a set (or a distribution)

The training samples are merely representatives of a whole set of possible outputs (e.g. a manifold of outputs).

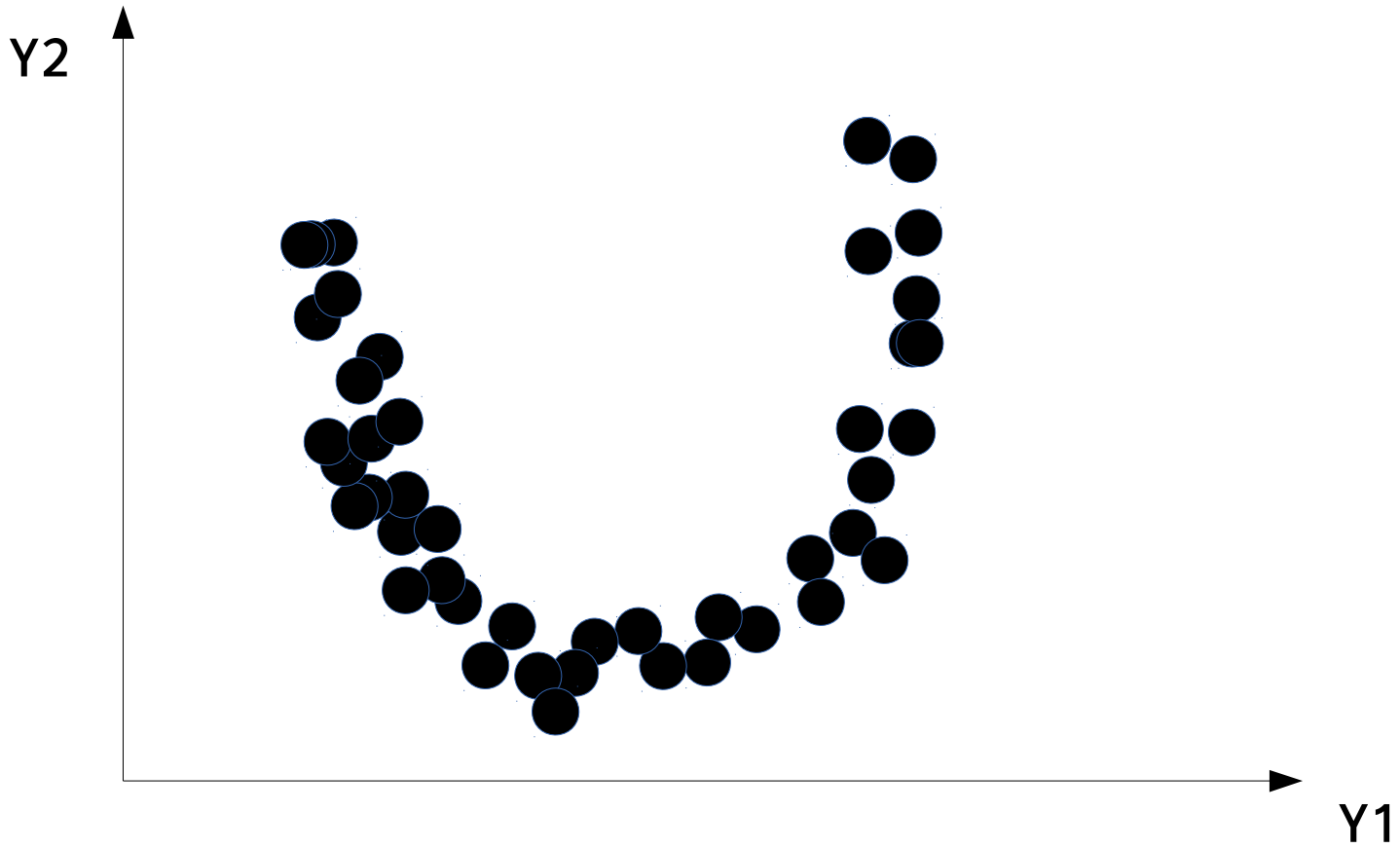
Question: how do we design a loss function so that the machine is enticed to output points on the data manifold, but not punished for producing points different from the desired output in the training set?




Energy-Based
Unsupervised Learning

Energy-Based Unsupervised Learning

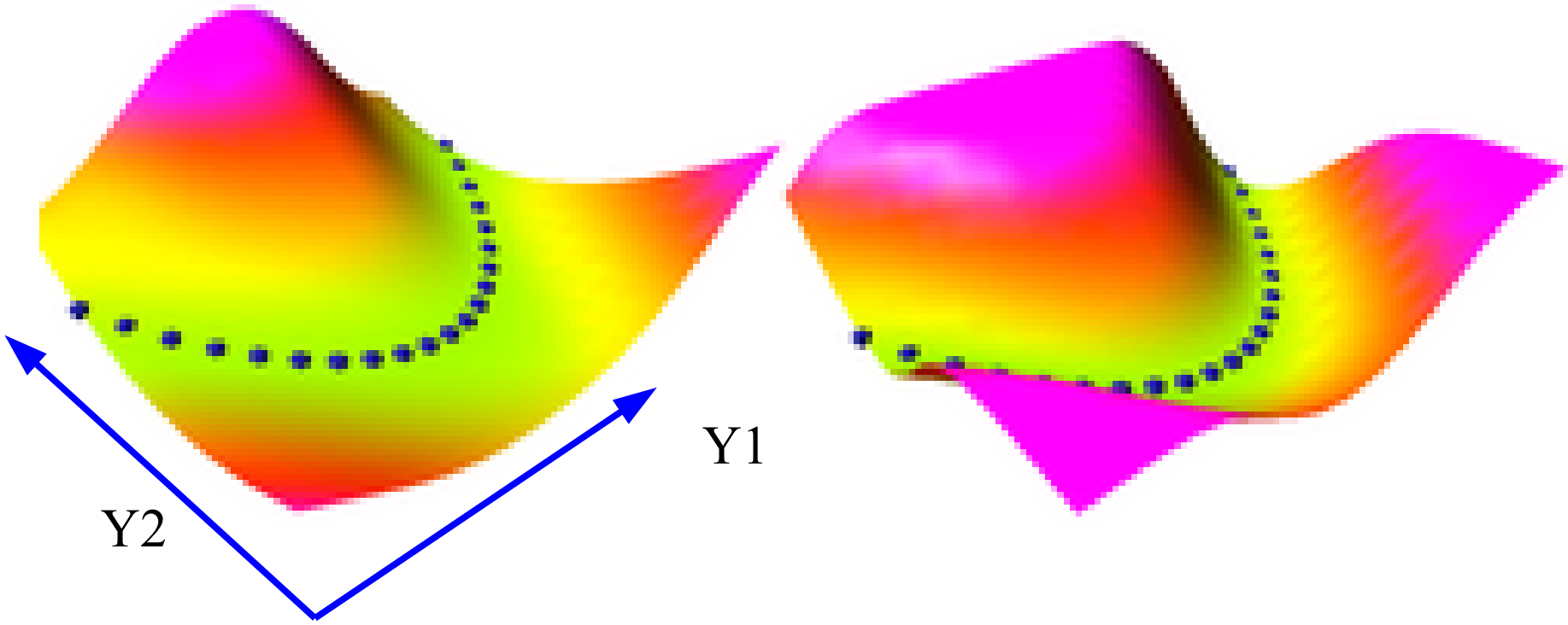
- Learning an energy function (or contrast function) that takes
  - Low values on the data manifold
  - Higher values everywhere else



Capturing Dependencies Between Variables with an Energy Function

- The energy surface is a "contrast function" that takes low values on the data manifold, and higher values everywhere else
  - Special case: energy = negative log density
  - Example: the samples live in the manifold

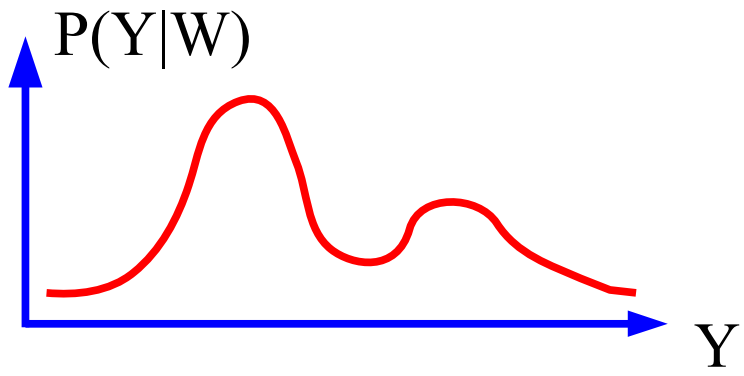
$$Y_2 = (Y_1)^2$$



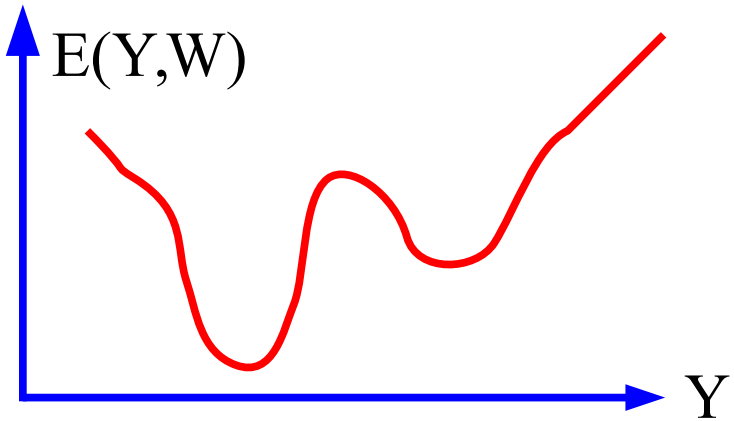
Transforming Energies into Probabilities (if necessary)

- The energy can be interpreted as an unnormalized negative log density
- Gibbs distribution: Probability proportional to exp(-energy)
  - Beta parameter is akin to an inverse temperature
- Don't compute probabilities unless you absolutely have to
  - Because the denominator is often intractable

$$P(Y|W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}}$$

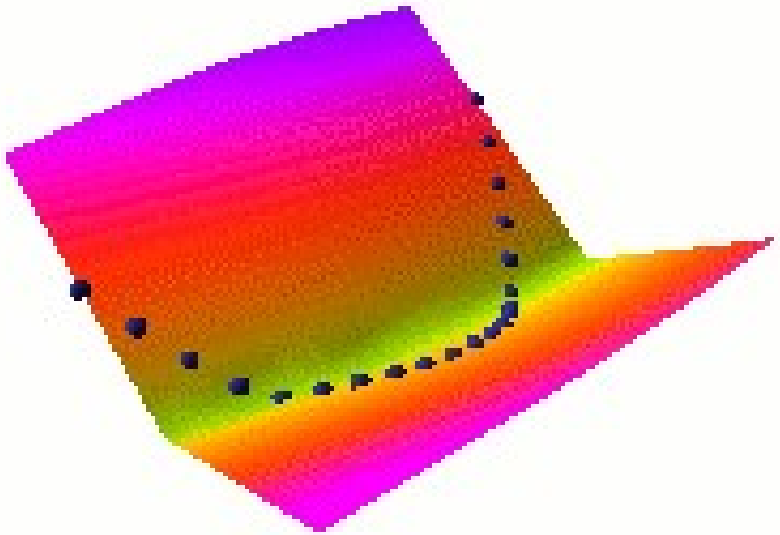


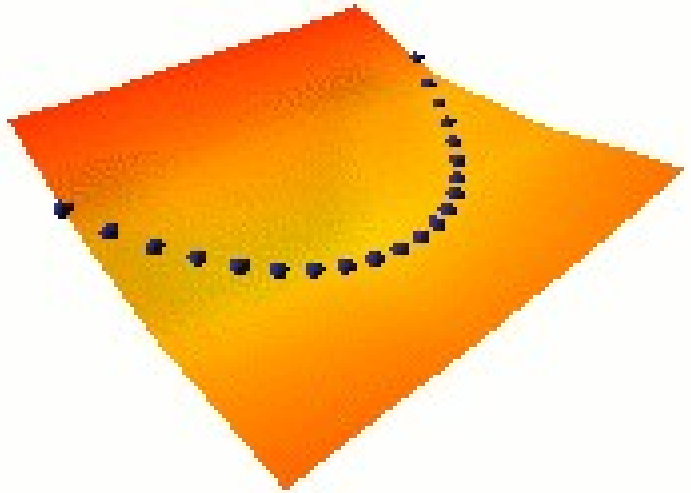
$$E(Y,W) \propto -\log P(Y|W)$$



Learning the Energy Function

- parameterized energy function E(Y,W)
  - Make the energy low on the samples
  - Make the energy higher everywhere else
  - Making the energy low on the samples is easy
  - But how do we make it higher everywhere else?





Eight Strategies to Shape the Energy Function

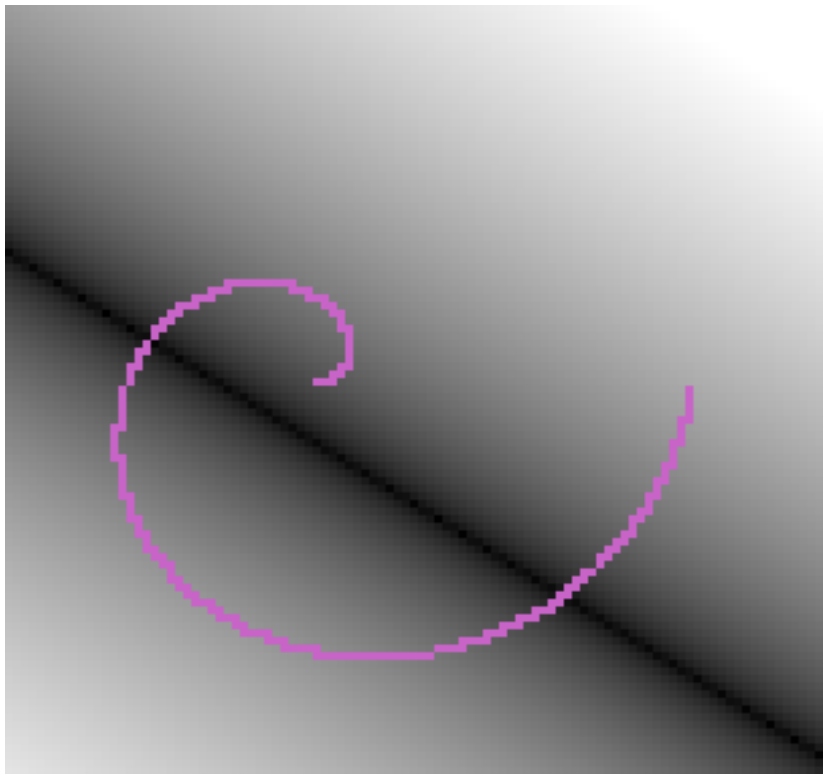
- 1. build the machine so that the volume of low energy stuff is constant
  - PCA, K-means, GMM, square ICA
- 2. push down of the energy of data points, push up everywhere else
  - Max likelihood (needs tractable partition function)
- 3. push down of the energy of data points, push up on chosen locations
  - contrastive divergence, Ratio Matching, Noise Contrastive Estimation, Minimum Probability Flow
- 4. minimize the gradient and maximize the curvature around data points
  - score matching
- 5. train a dynamical system so that the dynamics goes to the manifold
  - denoising auto-encoder
- 6. use a regularizer that limits the volume of space that has low energy
  - Sparse coding, sparse auto-encoder, PSD
- 7. if E(Y) = ||Y - G(Y)||^2, make G(Y) as "constant" as possible.
  - Contracting auto-encoder, saturating auto-encoder
- 8. Adversarial training: generator tries to fool real/synthetic classifier.

#1: constant volume of low energy

- 1. build the machine so that the volume of low energy stuff is constant
  - PCA, K-means, GMM, square ICA...

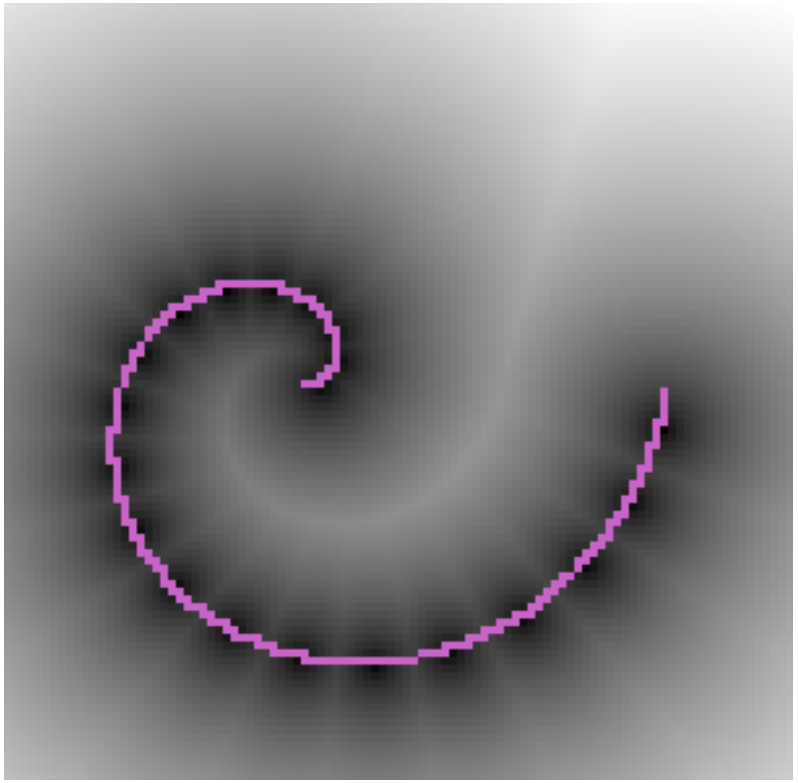
PCA

$$E(Y)=\|W^T WY - Y\|^2$$



K-Means,
Z constrained to 1-of-K code

$$E(Y)=min_z \sum_i \|Y - W_i Z_i\|^2$$



#2: push down of the energy of data points, push up everywhere else

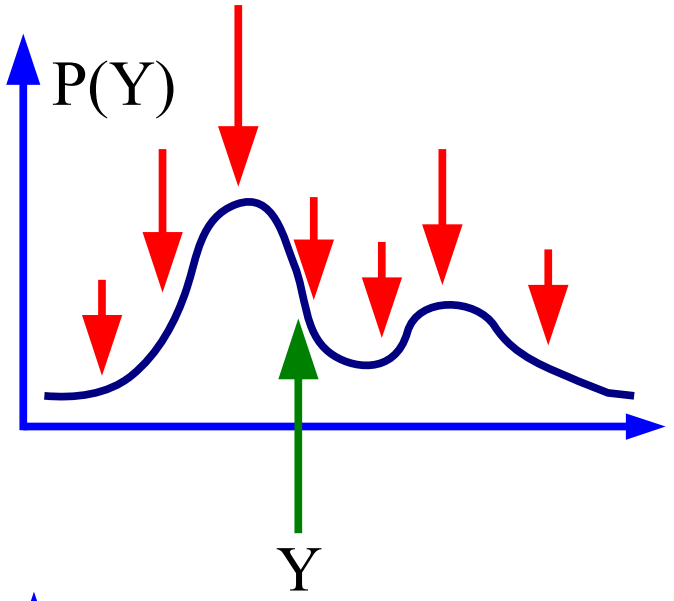
- Max likelihood (requires a tractable partition function)

Maximizing P(Y|W) on training samples

make this big

$$P(Y|W) = \frac{e^{-\beta E(Y,W)}}{\int_y e^{-\beta E(y,W)}}$$

make this small

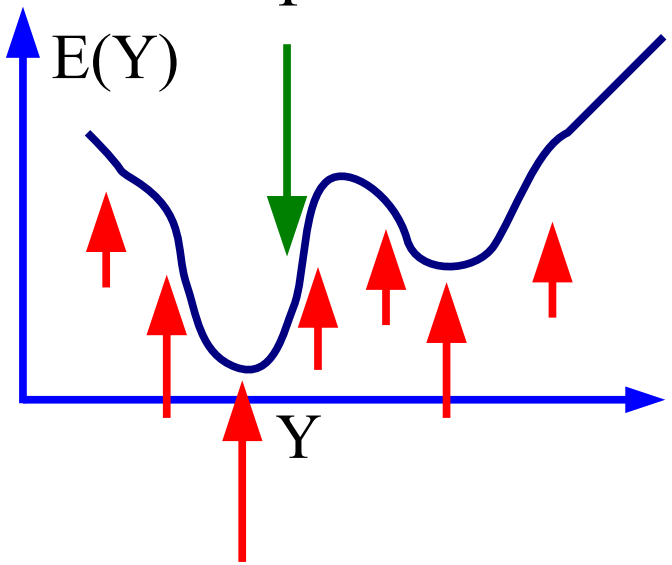


Minimizing -log P(Y,W) on training samples

$$L(Y,W) = E(Y,W) + \frac{1}{\beta}\log\int_y e^{-\beta E(y,W)}$$

make this small

make this big



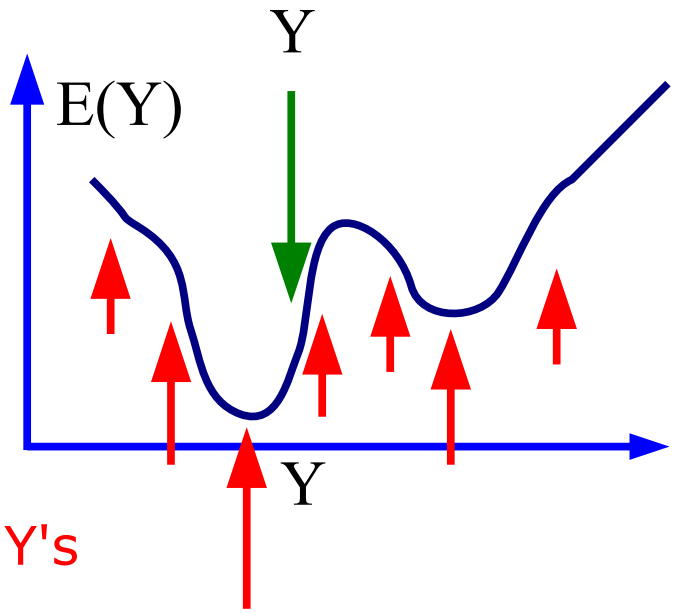
#2: push down of the energy of data points, push up everywhere else

Gradient of the negative log-likelihood loss for one sample Y:

$$\frac{\partial L(Y,W)}{\partial W} = \frac{\partial E(Y,W)}{\partial W} - \int_y P(y|W) \frac{\partial E(y,W)}{\partial W}$$

Gradient descent:

$$W \leftarrow W - \eta \frac{\partial L(Y,W)}{\partial W}$$



Pushes down on the energy of the samples

Pulls up on the energy of low-energy Y's

$$W \leftarrow W - \eta \frac{\partial E(Y,W)}{\partial W} + \eta \int_y P(y|W) \frac{\partial E(y,W)}{\partial W}$$

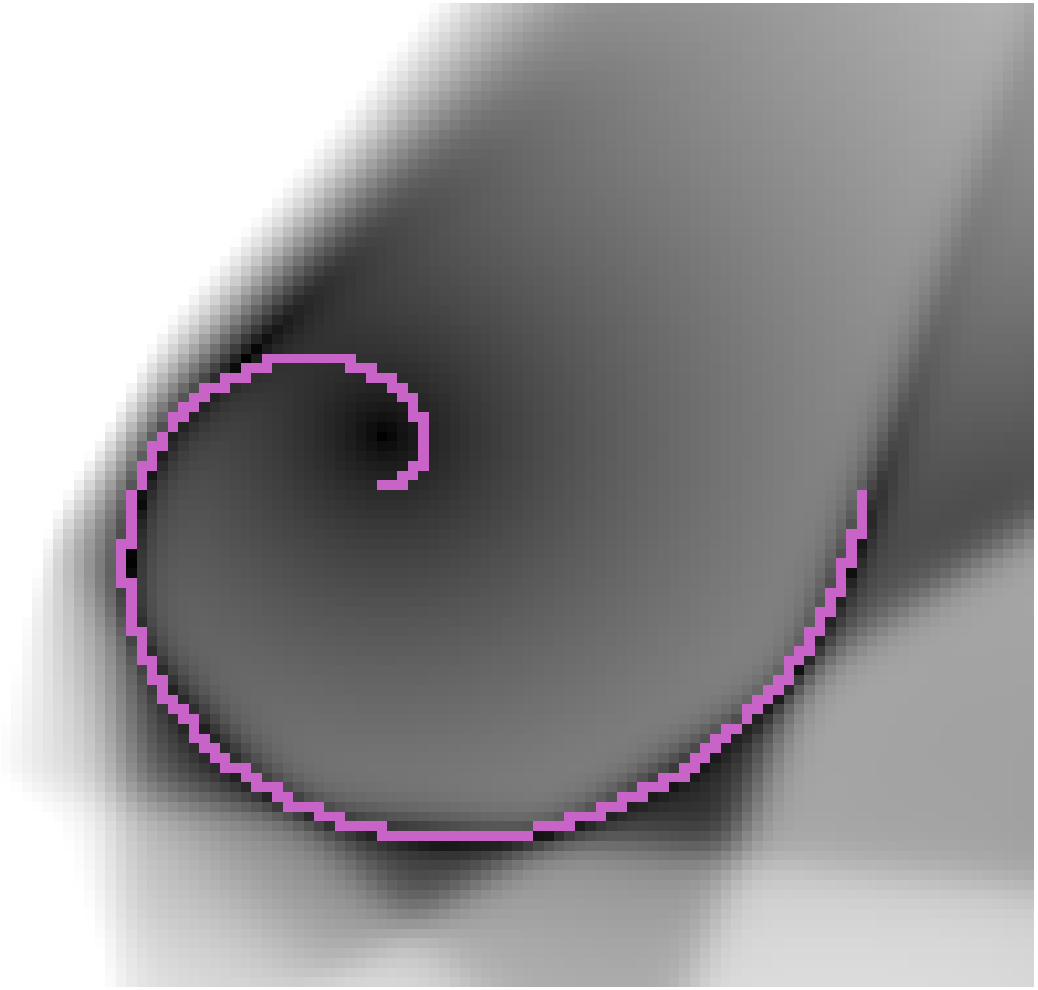
#3. push down of the energy of data points, push up on chosen locations

- contrastive divergence, Ratio Matching, Noise Contrastive Estimation, Minimum Probability Flow
- Contrastive divergence: basic idea
  - Pick a training sample, lower the energy at that point
  - From the sample, move down in the energy surface with noise
  - Stop after a while
  - Push up on the energy of the point where we stopped
  - This creates grooves in the energy surface around data manifolds
  - CD can be applied to any energy function (not just RBMs)
- Persistent CD: use a bunch of "particles" and remember their positions
  - Make them roll down the energy surface with noise
  - Push up on the energy wherever they are
  - Faster than CD
- RBM

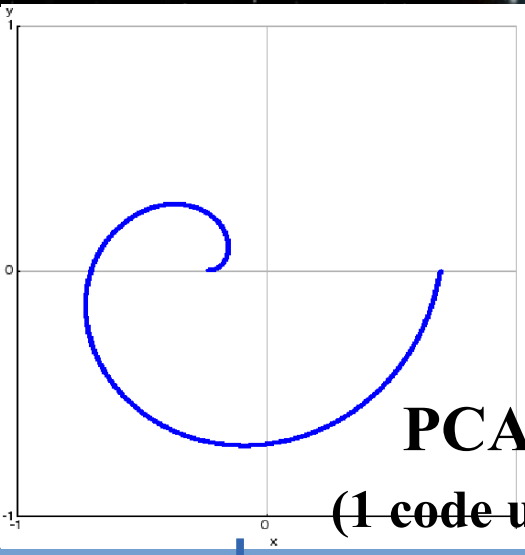
$$E(Y,Z) = -Z^T WY \qquad E(Y) = -\log \sum_z e^{Z^T WY}$$

#6. use a regularizer that limits the volume of space that has low energy

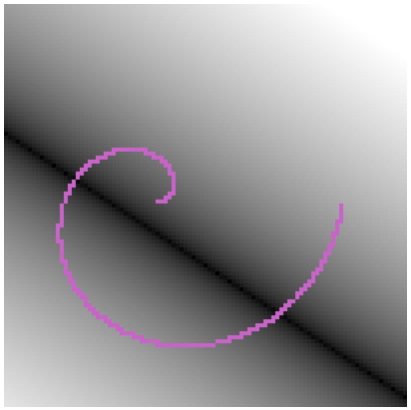
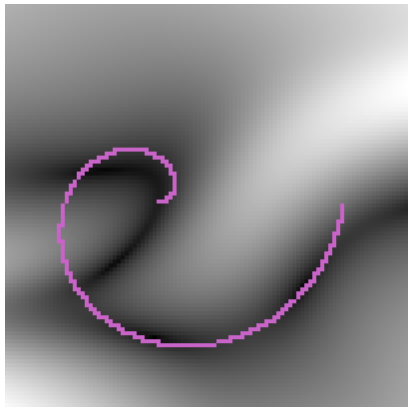
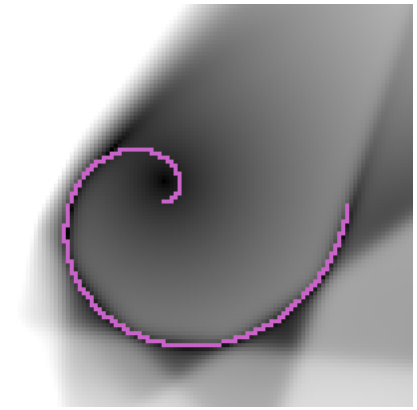
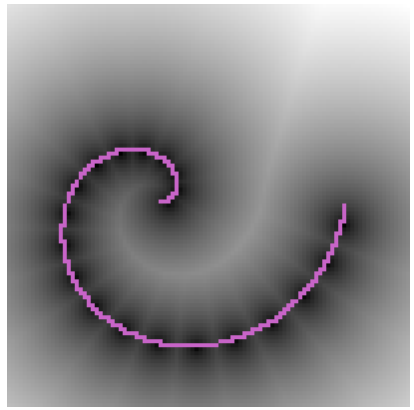
- Sparse coding, sparse auto-encoder, Predictive Saprse Decomposition

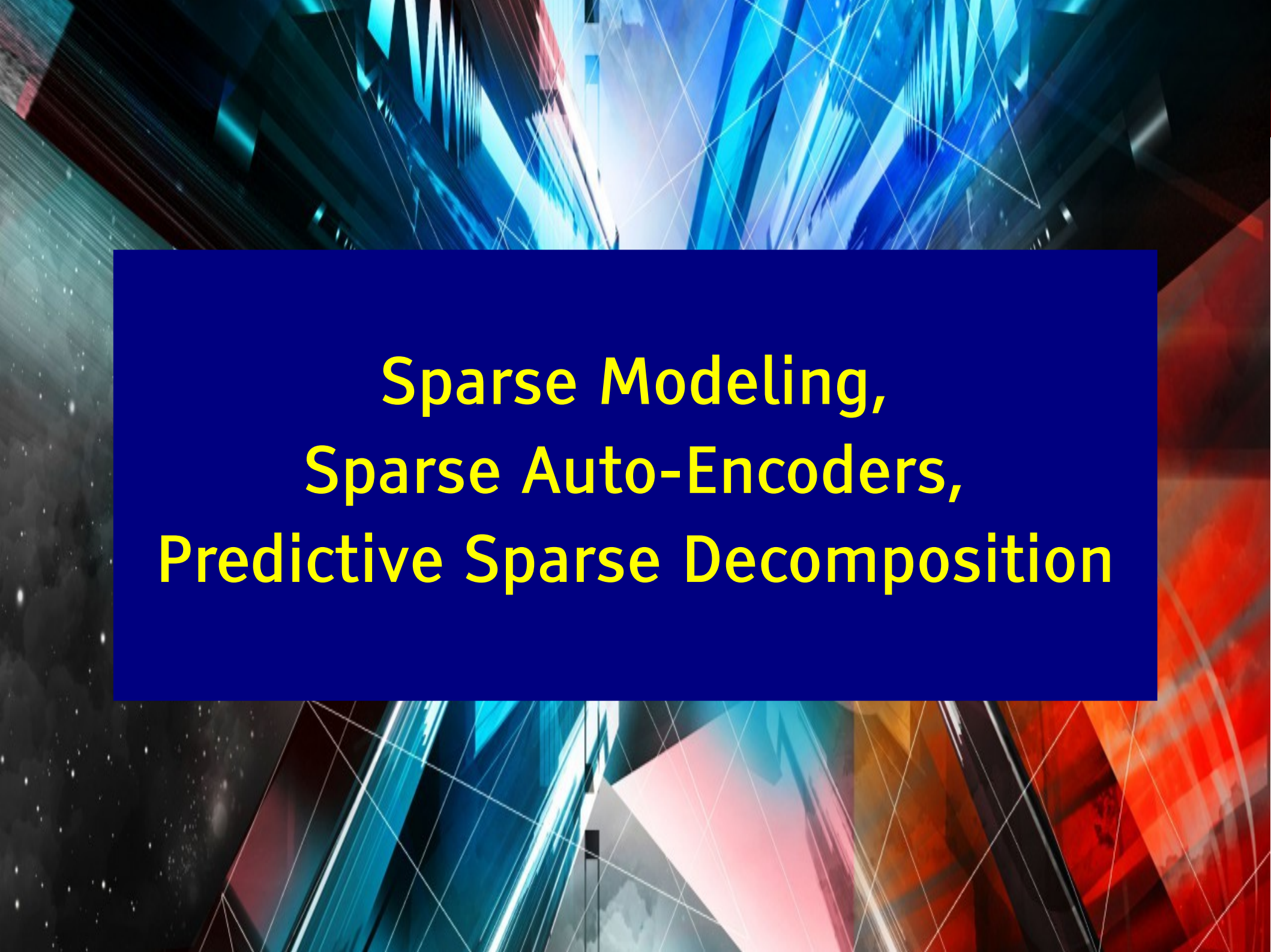


Energy Functions of Various Methods



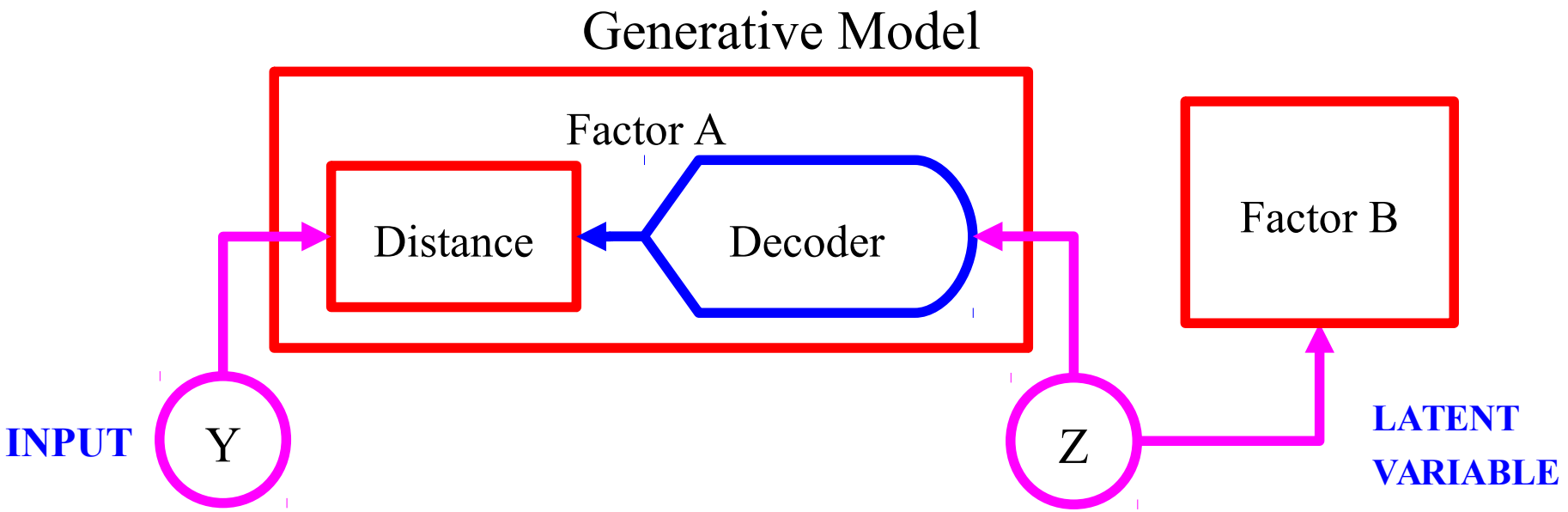
- 2 dimensional toy dataset: spiral
- Visualizing energy surface
  - (black = low, white = high)

	PCA (1 code unit)	autoencoder (1 code unit)	sparse coding (20 code units)	K-Means (20 code units)
encoder	$W'Y$	$\sigma(W_e Y)$	$\sigma(W_e Z)$	–
decoder	WZ	$W_d Z$	$W_d Z$	WZ
energy	$\|\|Y-WZ\|\|^2$	$\|\|Y-WZ\|\|^2$	$\|\|Y-WZ\|\|^2$	$\|\|Y-WZ\|\|^2$
loss	$F(Y)$	$F(Y)$	$F(Y)$	$F(Y)$
pull-up	dimens.	dimens.	sparsity	1-of-N code
				


Sparse Modeling,
Sparse Auto-Encoders,
Predictive Sparse Decomposition

How to Speed Up Inference in a Generative Model?

- Factor Graph with an asymmetric factor
- Inference Z → Y is easy
  - Run Z through deterministic decoder, and sample Y
- Inference Y → Z is hard, particularly if Decoder function is many-to-one
  - MAP: minimize sum of two factors with respect to Z
  - Z* = argmin_z Distance[Decoder(Z), Y] + FactorB(Z)
- Examples: K-Means (1of K), Sparse Coding (sparse), Factor Analysis

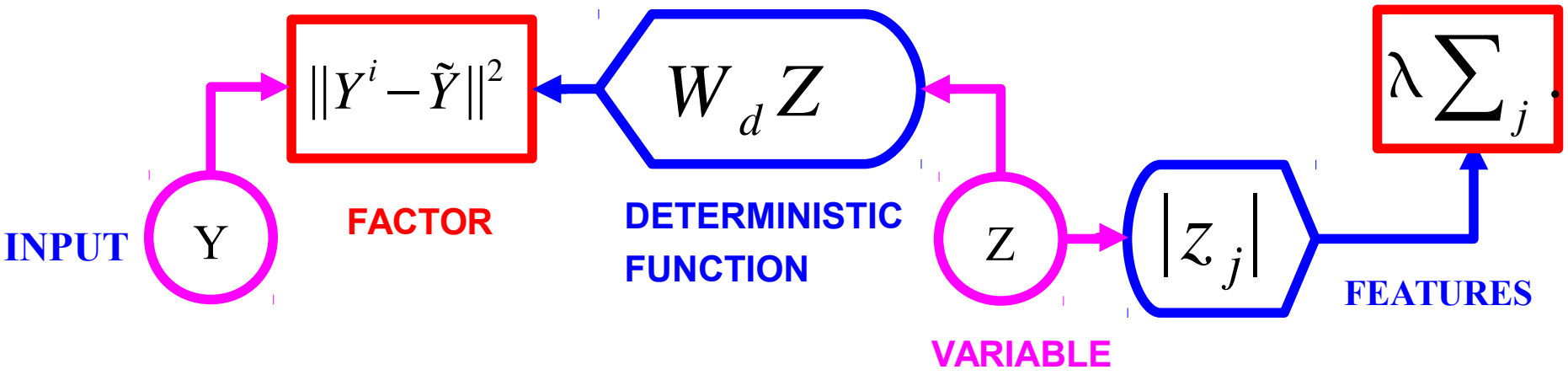


Sparse Coding & Sparse Modeling

[Olshausen & Field 1997]

- Sparse linear reconstruction
- Energy = reconstruction_error + code_prediction_error + code_sparsity

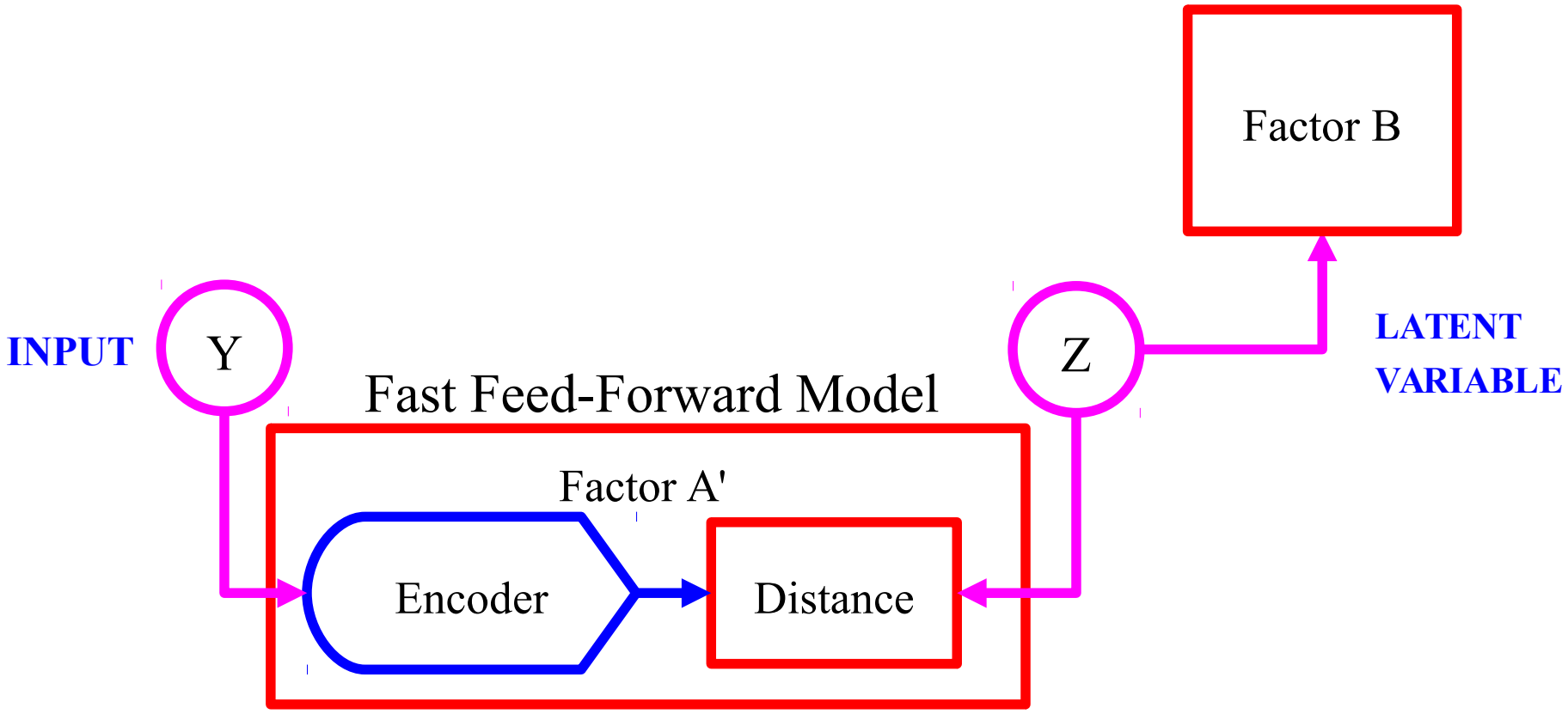
$$E(Y^i, Z) = \|Y^i - W_d Z\|^2 + \lambda \sum_j |z_j|$$



- Inference is slow $Y \rightarrow \hat{Z} = argmin_Z E(Y, Z)$

Encoder Architecture

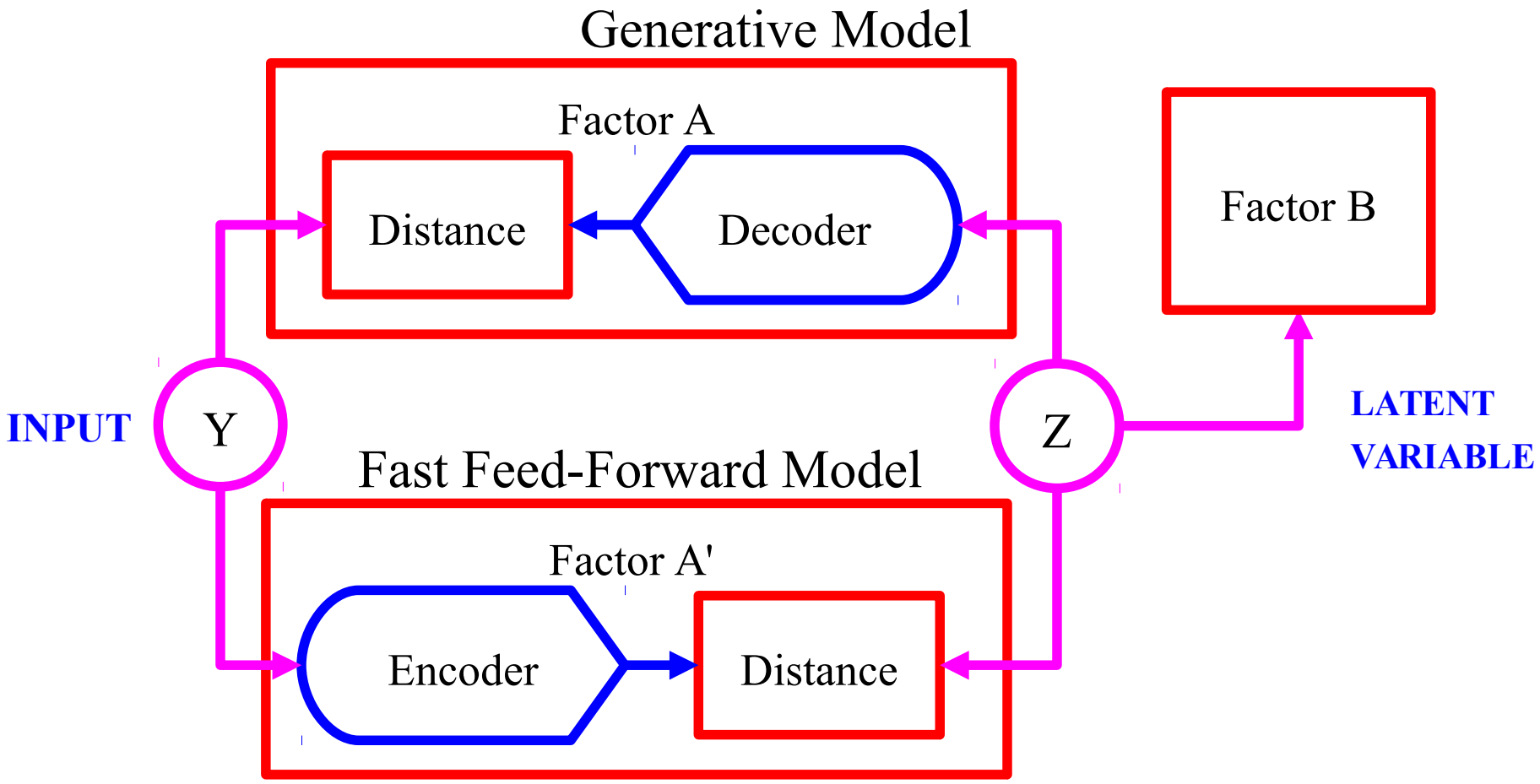
Examples: most ICA models, Product of Experts



Encoder-Decoder Architecture

[Kavukcuoglu, Ranzato, LeCun, rejected by every conference, 2008-2009]

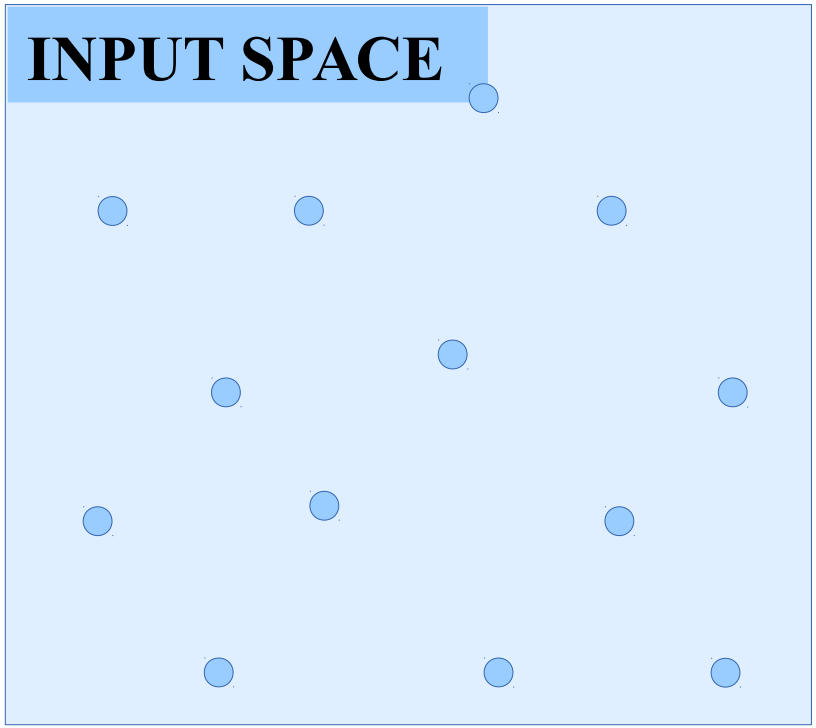
- Train a "simple" feed-forward function to predict the result of a complex optimization on the data points of interest

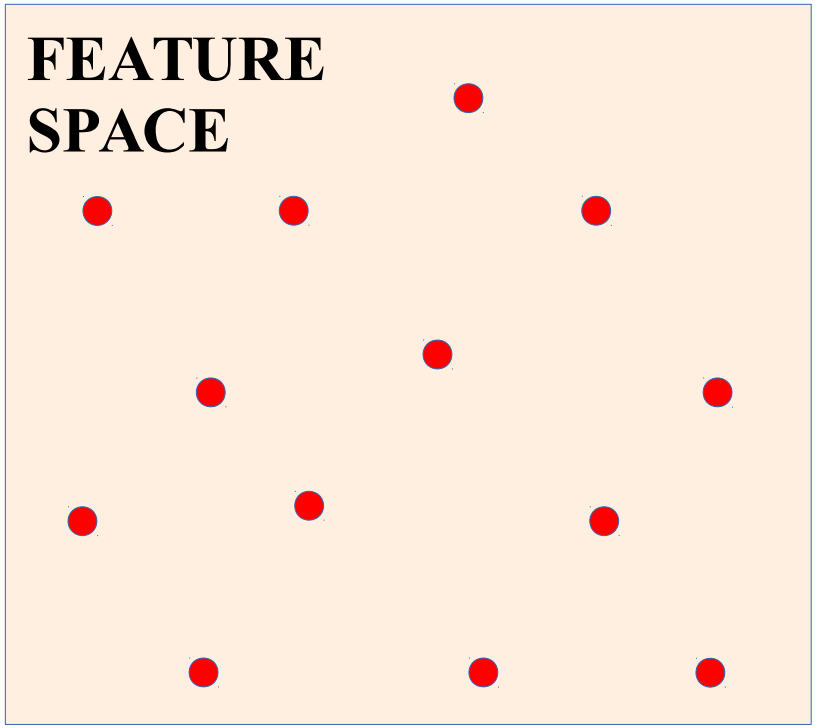


- 1. Find optimal Zi for all Yi; 2. Train Encoder to predict Zi from Yi

Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

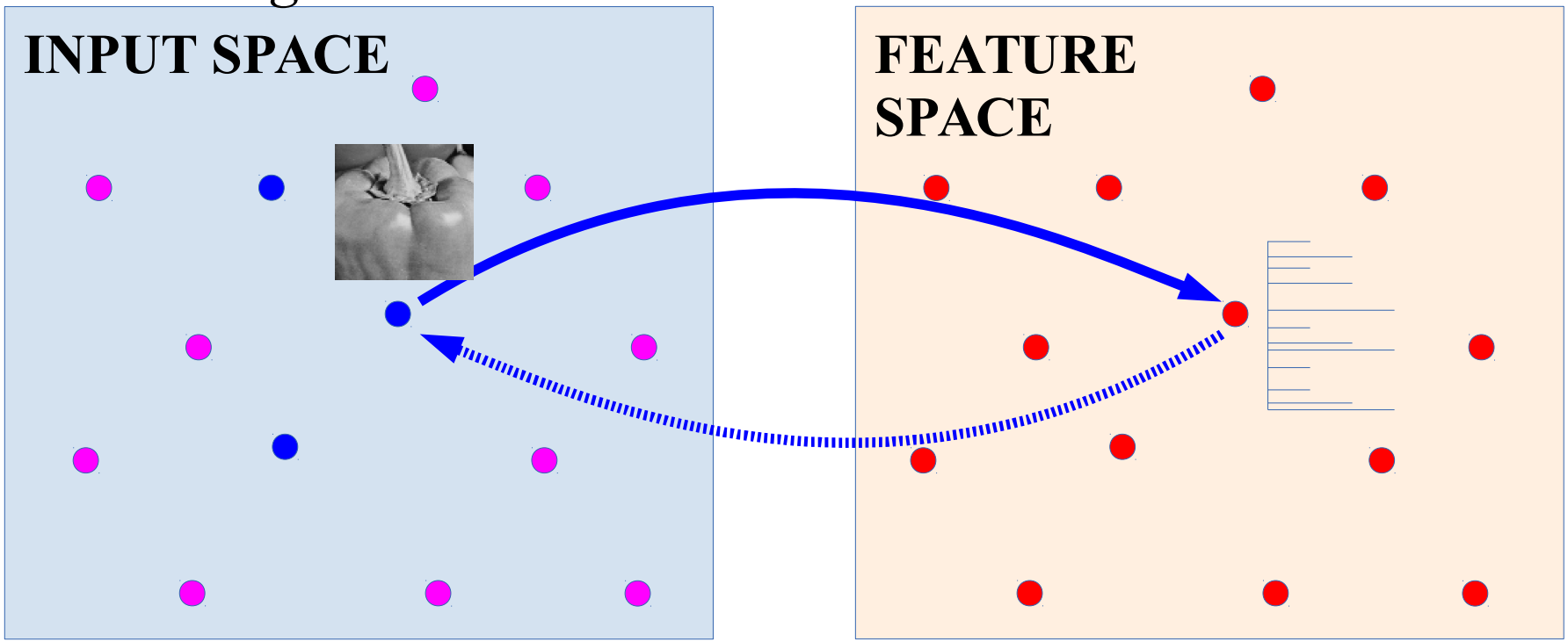




Why Limit the Information Content of the Code?

- **Training sample**
- **Input vector which is NOT a training sample**
- **Feature vector**

Training based on minimizing the reconstruction error over the training set

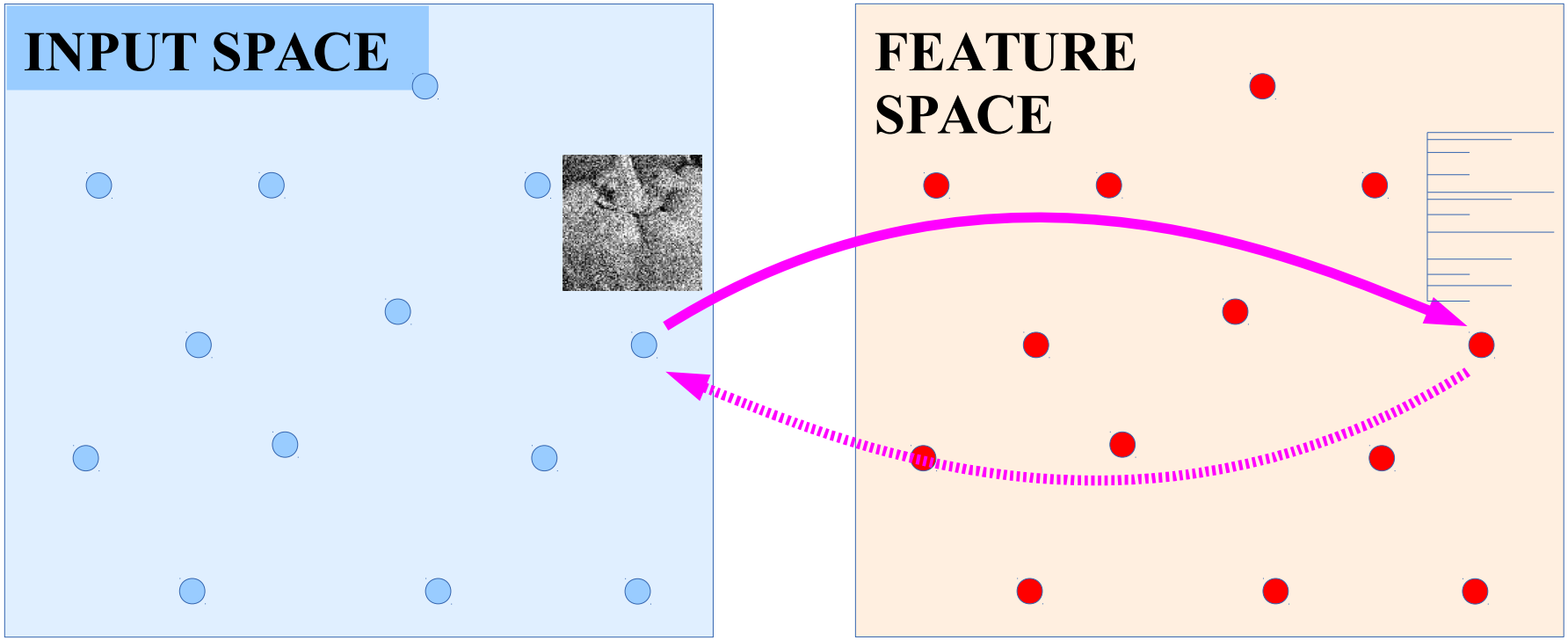


Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

BAD: machine does not learn structure from training data!!

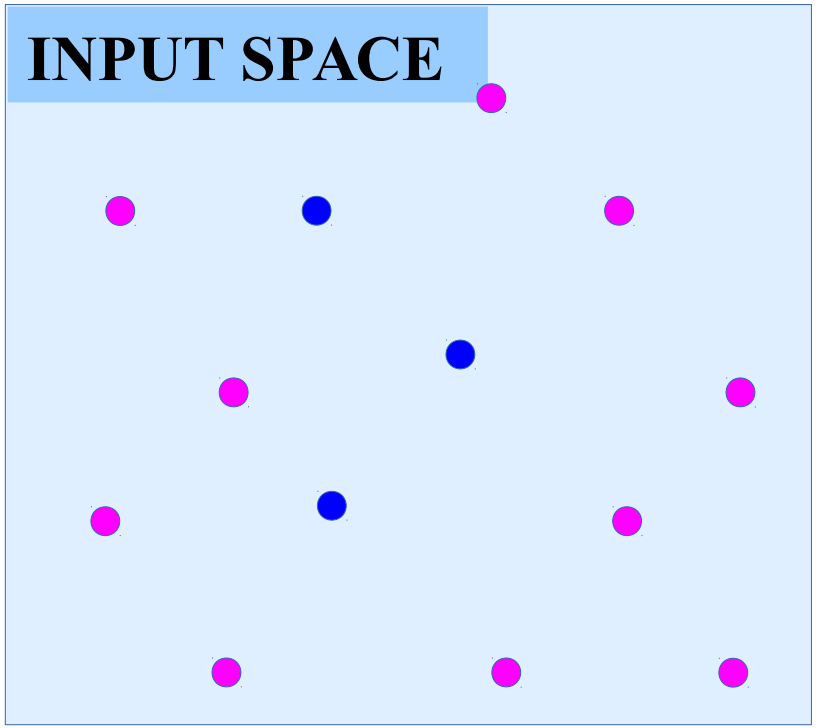
It just copies the data.

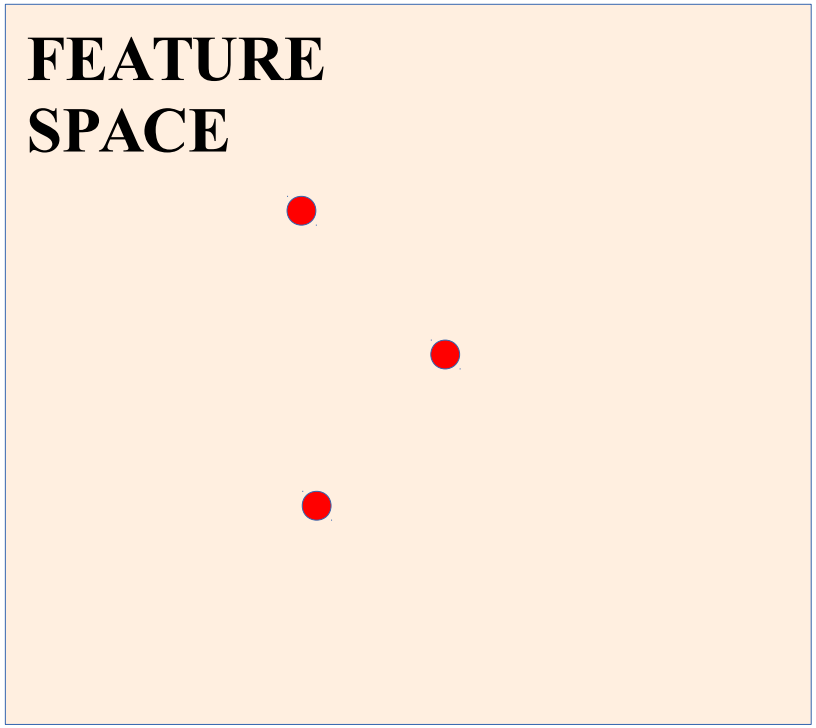


Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

IDEA: reduce number of available codes.

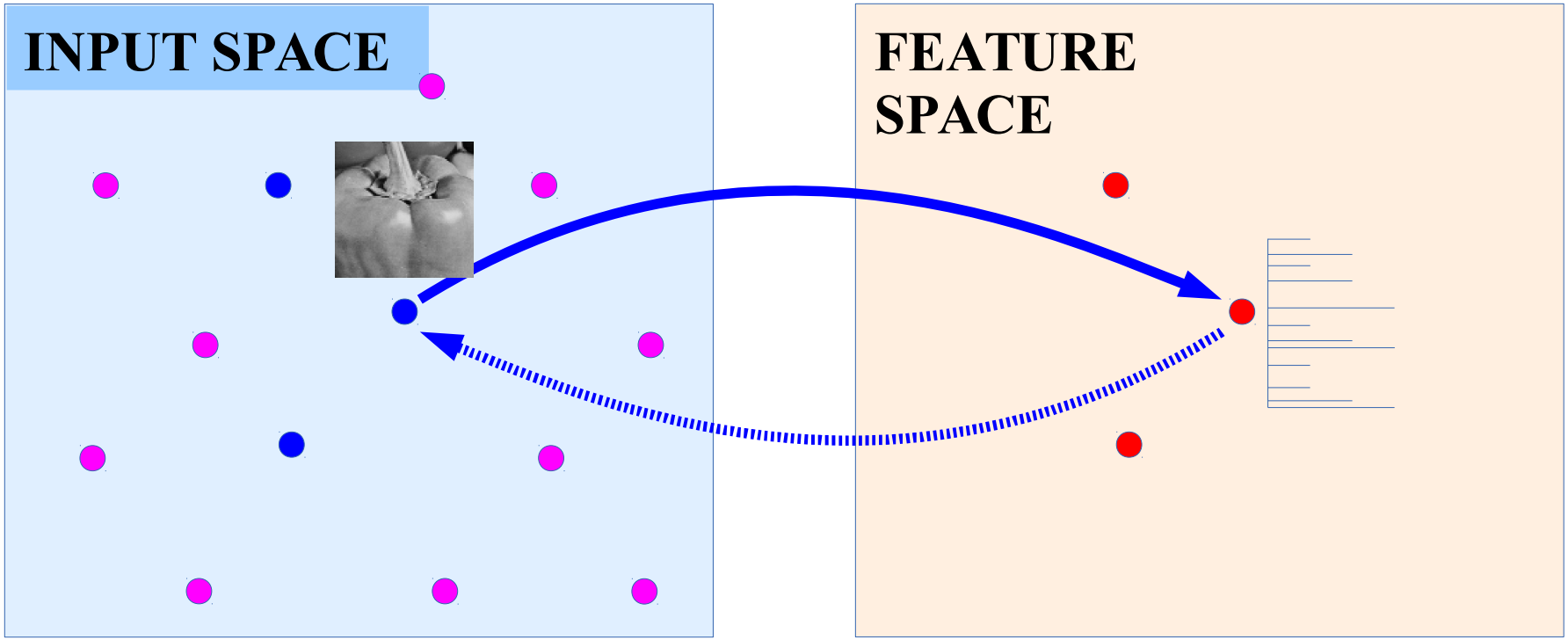




Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

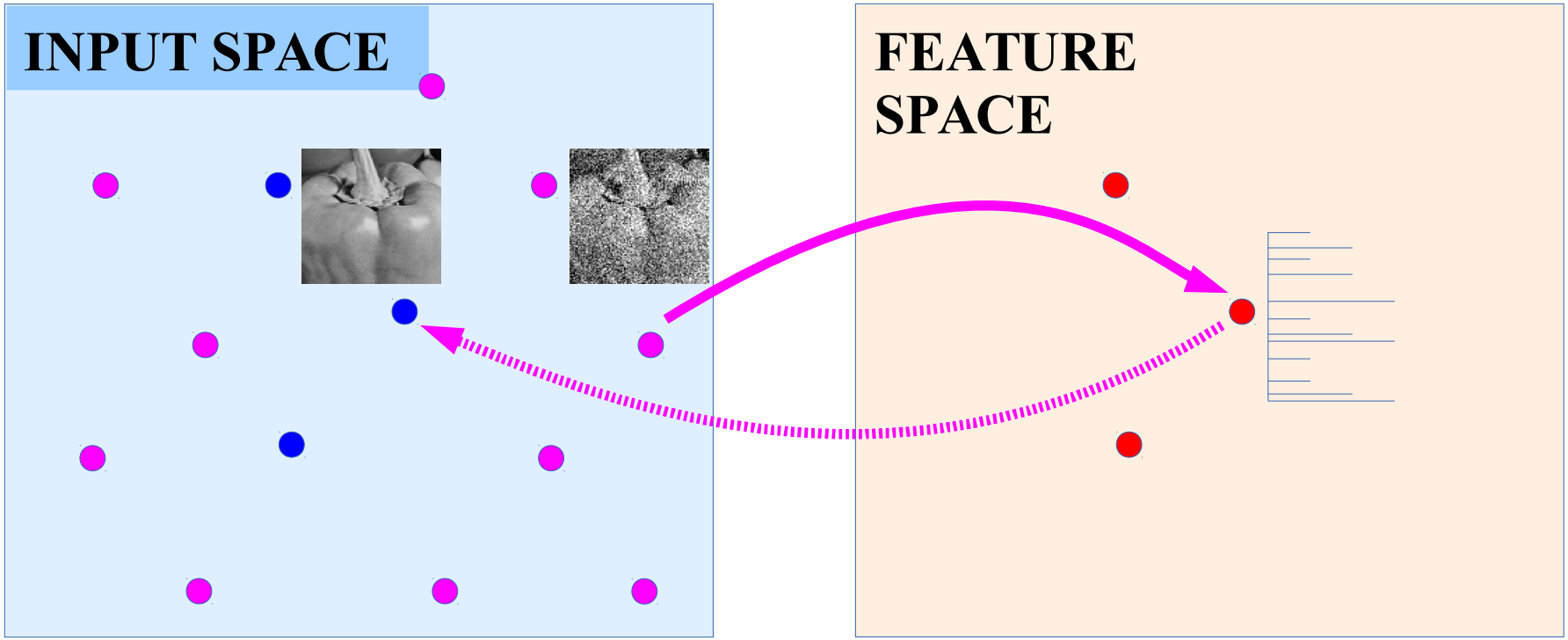
IDEA: reduce number of available codes.



Why Limit the Information Content of the Code?

- Training sample
- Input vector which is NOT a training sample
- Feature vector

IDEA: reduce number of available codes.



Predictive Sparse Decomposition (PSD): sparse auto-encoder

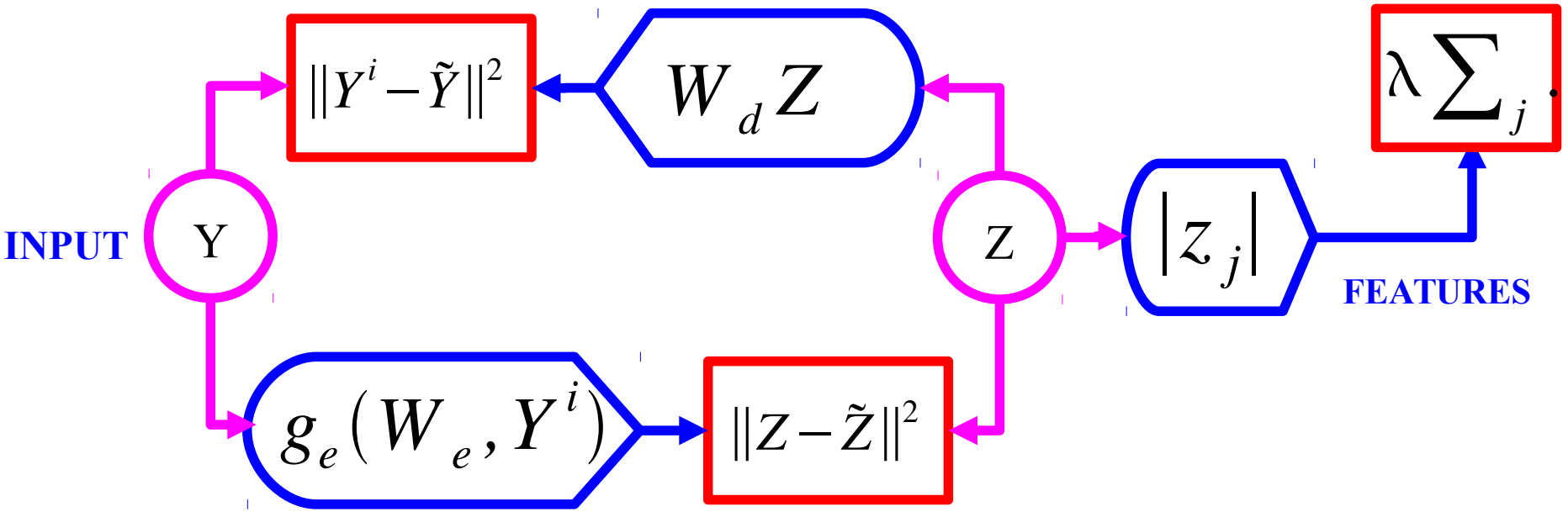
[Kavukcuoglu, Ranzato, LeCun, 2008 → arXiv:1010.3467],

Prediction the optimal code with a trained encoder

Energy = reconstruction_error + code_prediction_error + code_sparsity

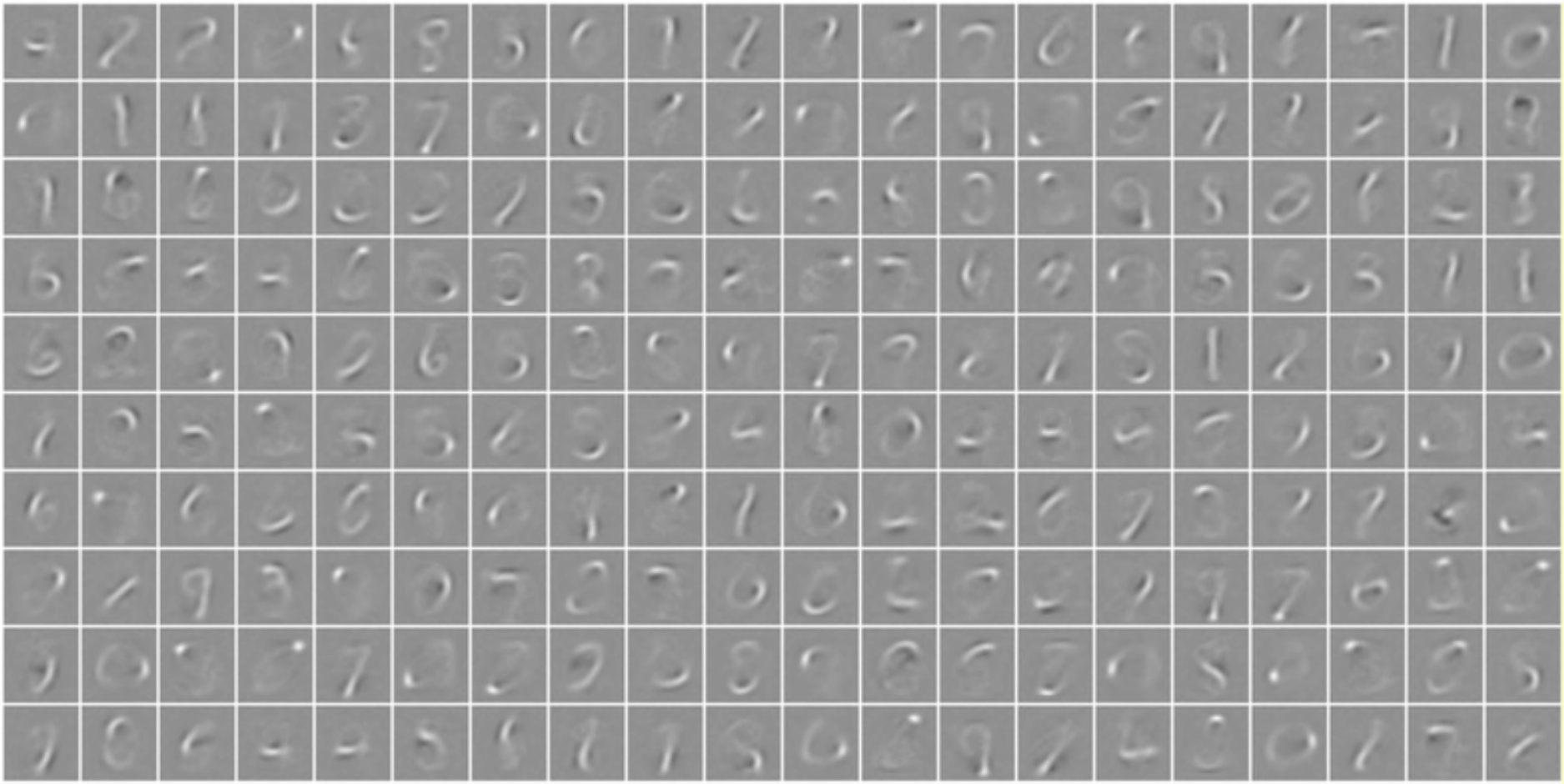
$$E(Y^i, Z) = \|Y^i - W_d Z\|^2 + \|Z - g_e(W_e, Y^i)\|^2 + \lambda \sum_j |z_j|$$

$$g_e(W_e, Y^i) = shrinkage(W_e Y^i)$$



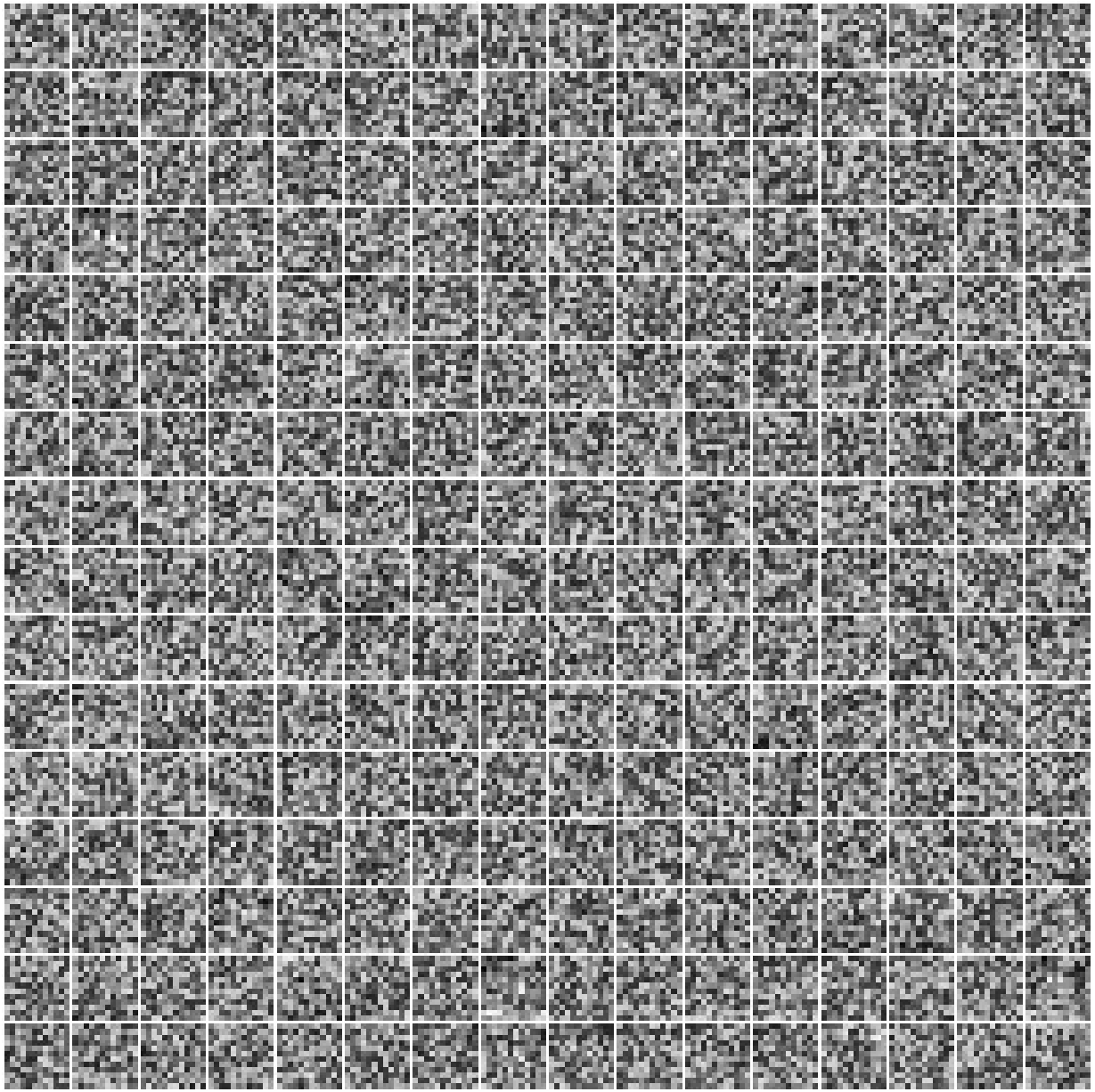
PSD: Basis Functions on MNIST

- Basis functions (and encoder matrix) are digit parts



Predictive Sparse Decomposition (PSD): Training

- Training on natural images patches.
  - 12X12
  - 256 basis functions



iteration no 0

Learned Features on natural patches: V1-like receptive fields